Improvements for Gate-Hiding Garbled Circuits*

Mike Rosulek†

October 4, 2017

Abstract

Garbled circuits have been highly optimized for practice over the last several years. Today's most efficient constructions treat different types of gates (*e.g.*, AND vs XOR) differently; as such, they leak the type of each gate. In many applications of garbled circuits, the circuit itself is public, so such leakage is tolerable. In other settings, however, it is desirable to hide the type of each gate.

In this paper we consider optimizing garbled circuits for the gate-hiding case. We observe that the best state-of-the-art constructions support only a limited class of gate functions, which turns out to undermine their improvements in several settings. These state-of-the-art constructions also require a non-minimal hardness assumption.

We introduce two new gate-hiding constructions of garbled circuits. Both constructions achieve the same communication complexity as the best state-of-the-art schemes, but support a more useful class of boolean gates and use only the minimal assumption of a secure PRF.

1 Introduction

Garbled circuits were first proposed by Yao in the 1980s [Yao82] and have since become the target of many improvements. Garbled circuits form the basis of secure two-party computation protocols and many other applications in cryptography.

In a typical scenario involving two-party computation, both parties agree on some circuit f that they wish to evaluate. Since f is public, the garbled circuits in these protocols do not need to hide anything about f; they need to hide only the *inputs* to f. However, in some applications like private function evaluation (PFE) [AF90, KM11, MS13] it is useful for the garbled circuit to hide information about the circuit itself.

In this work, we study garbled circuit constructions that are **gate-hiding** — that is, they leak only the topology of the circuit, while hiding the type of each gate (e.g., AND, XOR, NOR).

Comparing efficiency of garbling schemes. With the ubiquity of native AES instructions on modern CPUs, applications of garbled circuits are rarely CPU-bound but are usually network-bound (cf. [ZRE15]). Hence, the most important metric for measuring the efficiency of garbled circuits is their **size** (typically bits per gate). In all of the schemes we will discuss, the difference in garbled circuit size is only in the constant factors. All these schemes produce garbled gates with size $c\lambda + d$, where c and d are small constants and λ is the computational security parameter.

*Full version of a paper appearing in Indocrypt 2017

†Oregon State University, rosulekm@eecs.oregonstate.edu. Partially supported by NSF awards 1149647 & 1617197.

	size (bits)	gate basis	garble cost		eval. cost		assump.
			H	interp	H	interp	
Textbook Yao	$4\lambda + 4$	$\mathcal{G}_{\text{all}}$	4	0	1	0	PRF
GRR3 [NPS99]	$3\lambda + 4$	$\mathcal{G}_{\text{all}}$	4	0	1	0	PRF
KKS [KKS16]	$2\lambda + 8$	$\mathcal{G}_{\text{symm}}$	3	0	1	0	circ+RK
WM [WM17]	$2\lambda + 2$	$\mathcal{G}_{\text{symm}}$	3	1	1	1	circ+RK
KKS$^{+\text{unary}}$ (§3)	$3\lambda + 8$	$\mathcal{G}_{\text{all}}$	5	0	2	0	circ+RK
WM$^{+\text{unary}}$ (§3)	$3\lambda + 2$	$\mathcal{G}_{\text{all}}$	5	1	2	1	circ+RK
new (§4)	$2\lambda + 4$	$\mathcal{G}_{\text{non-const}}$	4	2	1	1	PRF
new (§5)	$2\lambda + 8$	$\mathcal{G}_{\text{non-const}}$	4	0	1	0	PRF

Figure 1: Comparison of gate-hiding garbling schemes. All costs are listed per-gate. "H" refers to calls to a symmetric-key primitive (see Section 2.3 for the required primitive). "interp" refers to interpolations of degree-2 polynomials over $GF(2^\lambda)$. "circ+RK" refers to a circularity+related-key assumption on H.

1.1 State of the Art

It is folklore that the "textbook" Yao garbling scheme is gate-hiding. Indeed, in the security proof for Yao's protocol from [LP09], there is a hybrid in which each garbled gate is replaced by a garbled "always-zero" gate. Since every garbled gate is indistinguishable from such a constant gate, the scheme hides the type of gate. The same property holds for simple improvements like *garbled row reduction* (GRR3) [NPS99]; that is, they hide the type of each gate. These two schemes garble each gate at a cost of $4\lambda + 4$ and $3\lambda + 4$ bits, respectively.[1]

The most efficient constructions of garbled circuits are derived from the Free XOR optimization [KS08] (including [KMR14, ZRE15, GLNP15]). These constructions are **not** gate-hiding because the evaluator must behave very differently for XOR gates vs. non-XOR gates. This is what allows XOR gates to be garbled more efficiently than other gates in these constructions. These schemes therefore leak whether a gate is XOR or not, while typically hiding all further distinctions.

Two recent papers, one by Kempka, Kikuchi, & Suzuki [KKS16] (hereafter KKS) and one by Wang & Malluhi [WM17] (hereafter WM), each describe a gate-hiding garbling scheme in which each garbled gate costs only $2\lambda + O(1)$ bits. Both schemes use the same representation of truth values as garbled wire labels, but otherwise use very different techniques for garbled gates.[2] These two schemes are currently the most lightweight gate-hiding schemes.[3] We discuss them in more detail in Section 3.

A summary of the state of the art for gate-hiding garbling schemes is given in Figure 1.

1.2 What Kinds of Gates are Supported, and Why Does It Matter?

Closer inspection of Figure 1 reveals that these constructions are not entirely interchangeable. In particular, *they support different classes of gate functionalities.* We identify three different classes of gate functionalities below:

[1] The "extra" 4 bits come from using the point-permute optimization. In practice one would typically use the underlying cryptographic primitive (*e.g.*, AES) in a way that gives security $\lambda = 127$, and then the garbled gates become a clean multiple of 128 bits in length. All of the constructions in this work use the point-permute optimization, which we discuss in greater detail in Section 2.4.

[2] KKS also show how to reduce the size of garbled gates at the input layer of a circuit. In this work we focus on internal gates of a circuit, and assume that the input gates represent only a small fraction of the circuit.

[3] In this paper we restrict our attention to constructions based on symmetric-key primitives only. There exist garbled circuit constructions based on very expensive primitives (functional encryption, FHE) where the cost of every garbled gate is constant.

$\mathcal{G}_{\text{all}}$: all gates $g : \{0,1\}^2 \to \{0,1\}$

$\mathcal{G}_{\text{symm}}$: all gates satisfying $g(0,1) = g(1,0)$

$\mathcal{G}_{\text{non-const}}$: all gates except for the degenerate ones $(a,b) \mapsto 0$ and $(a,b) \mapsto 1$

As mentioned above, it is not hard to see that textbook Yao and GRR3 support $\mathcal{G}_{\text{all}}$-gates and are gate-hiding with respect to this class. That is, they can garble literally any boolean gate functionality, in a way that hides the choice of gate.

However, the KKS and WM schemes — the schemes with smallest garbled gate size — **support only $\mathcal{G}_{\text{symm}}$-gates.** While this seems like a minor limitation, we point out two reasons it can be problematic:

NOT gates. When dealing with fan-in-2 gates, one can usually think of any unary NOT gates being "absorbed" into all downstream (in the direction of evaluation) gates.[4] This leads to non-symmetric gates like $(a,b) \mapsto a \wedge \overline{b}$. Almost all garbling schemes support "absorbed" negations at no additional cost, and in a way that hides the presence of the negations. Unfortunately, the glaring exceptions to this rule are in fact the KKS and WM schemes, which cannot garble these non-symmetric gates at all (we elaborate in Section 3)!

This raises the question of how to deal with a circuit containing NOT gates. While it is possible to express any circuit just in terms of NAND gates (which are symmetric), one will obtain smaller circuits by using a larger class of gates. Indeed, most of the available boolean circuits used for MPC are expressed as AND/XOR/NOT gates [TS, HS]. In Section 3 we argue that NOT gates inherently have extra cost in the KKS & WM schemes, due to the structure of wire labels in these schemes. In contrast, a garbling scheme that is gate-hiding for $\mathcal{G}_{\text{non-const}}$ would not suffer from the same limitation, since this class of gates is closed under "absorbed" NOT gates.

In short, closure under negating individual gate-inputs is a useful property, since it implies that NOT gates will always be free. But schemes that support only $\mathcal{G}_{\text{symm}}$ do not have this property.

Multiplexers, Pass-through gates. A multiplexer has three inputs and computes $(x_0, x_1, s) \mapsto x_s$. A pass-through gate is simply a multiplexer whose selection input s has been fixed, corresponding to either $(a,b) \mapsto a$ or $(a,b) \mapsto b$. These kinds of gates are clearly not in $\mathcal{G}_{\text{symm}}$, but are vital in most applications of gate-hiding garbled circuits. Garbling a pass-through gate in a gate-hiding way is therefore equivalent to garbling a multiplexer with its selection bit secretly fixed by the garbler. This is an approach used by Paus *et al.* [PSS09] in an application to semi-private function evaluation, where the function being evaluated is hidden within a known class of functions. Constructions of *universal circuits* [Val76, LMS16, KS16, GAKS17] likewise use significant amount of multiplexers, and when the garbler is the party who programs the universal circuit, the multiplexer is meant to be replaced by a pass-through gate. Other variants of universal circuits [KKW16] explicitly use pass-through gates as a fundamental concept in their constructions.

1.3 Hardness Assumptions

In the non-gate-hiding setting, the best garbling schemes use the Free XOR optimization [KS08] to eliminate communication for XOR gates. Choi *et al.* [CKKZ12] showed that the Free XOR construction inherently relies on a nonstandard assumption: namely, that the underlying symmetric-key primitive have circular security and related-key security. For comparison, the minimal hardness assumption for garbled circuits is the existence of a PRF (equivalently, the existence of a one-way function). The best known way

[4] Absorbing the NOT gate into its "upstream" gate may not always work, since the upstream gate may have multiple fan-out.

for garbling XOR gates from standard assumptions is due to Gueron *et al.* [GLNP15], who garble XOR gates at a cost of λ bits each, using a standard PRF.

It is reasonable to use a stronger assumption to achieve efficiency that we do not know how to achieve otherwise. In the case of non-gate-hiding garbled circuits, a nonstandard assumption allows XOR gates to be garbled for free.

However, in the gate-hiding case we cannot expect to garble XOR gates for free, since we cannot expect to garble *all* gates for free (for evidence, see the lower bound of Zahur *et al.* [ZRE15]). When XOR gates are not free, it is not clear that a stronger hardness assumption is necessary. Yet the best existing gate-hiding schemes (KKS and WM) both rely on a free-XOR-like hardness assumption that involves correlated keys and circularity. A natural question is whether this flavor of assumption is necessary for highly efficient, gate-hiding garbled circuits.

1.4 Our Contributions

In this work we present three main results:

In Section 3 we show how to extend the KKS & WM constructions from a $\mathcal{G}_{\mathsf{symm}}$-scheme to a $\mathcal{G}_{\mathsf{all}}$-scheme. This comes at a price, however: the garbled gates must increase in size from $2\lambda + O(1)$ to $3\lambda + O(1)$, and evaluation requires an extra cryptographic operation. We give evidence that this extra cost is inevitable — i.e., the KKS & WM approaches cannot be extended beyond symmetric gates for free.

In Section 4 we revisit a scheme of Pinkas et al. [PSSW09] that garbles non-constant gates ($\mathcal{G}_{\mathsf{non\text{-}const}}$) for $2\lambda + 4$ bits. Their scheme, however, uses different methods to garble gates of even vs. odd parity (a gate has even parity if the number of 1's in its truth table is even). That is, the construction leaks the parity of a gate. We show that their odd-parity method can be adapted to work for even-parity gates as well, resulting in a gate-hiding scheme for $\mathcal{G}_{\mathsf{non\text{-}const}}$-gates with cost $2\lambda + 4$ bits per gate. While evaluation requires only a single cryptographic operation, it requires an additional polynomial interpolation step over $GF(2^\lambda)$, which in practice can cost roughly half of an AES evaluation (cf. [GLNP15]).

In Section 5 we present a new and novel gate-hiding scheme, inspired by a garbling technique of Gueron et al. [GLNP15]. This scheme supports $\mathcal{G}_{\mathsf{non\text{-}const}}$-gates, and results in garbled gates of size $2\lambda + 8$ bits. Evaluation involves just one cryptographic operation, and otherwise uses only XOR operations (in particular, no interpolation or finite field multiplications).

Our new constructions improve the concrete cost of applications of gate-hiding garbled circuits, by supporting circuits expressed over a more natural class of gates. Additionally, our new constructions require only the minimal hardness assumption of the existence of a PRF. As mentioned above, KKS & WM require a circularity/related-key assumption.

2 Preliminaries

2.1 Circuits

We represent circuits in the following way. All wires in the circuit (including input wires) are indexed in a topological ordering. For a circuit f, we define:

- $\mathsf{inputs}(f)$: the set of indices of input wires
- $\mathsf{gates}(f)$: the set of indices of non-input wires (i.e., wires that emanate from some internal gate)
- $\mathsf{outputs}(f)$: the set of indices of output wires (not necessarily disjoint from the other sets).

For each gate with index $i \in \mathsf{gates}(f)$, we define:

- $\mathsf{left}(i)$: the index of the gate's left input wire
- $\mathsf{right}(i)$: the index of the gate's right input wire
- $\mathsf{type}(i)$: the functionality of the gate; i.e., a function $g : \{0,1\}^2 \to \{0,1\}$

For a circuit f, we let $\mathsf{topo}(f)$ denote all of the above information except for $\mathsf{type}(i)$.

Let $\mathcal{G}$ be a set of boolean gates (e.g., $\mathcal{G}_{\mathsf{all}}$, $\mathcal{G}_{\mathsf{symm}}$). We say that a circuit f is a **$\mathcal{G}$-circuit** if $\mathsf{type}(i) \in \mathcal{G}$ for every $i \in \mathsf{gates}(f)$.

2.2 Garbled Circuits

We use the security definitions of Bellare et al. [BHR12]. A **garbling scheme** is a collection of algorithms $(\mathsf{Garble}, \mathsf{Encode}, \mathsf{Eval}, \mathsf{Decode})$, with the following semantics:

- $\mathsf{Garble}(1^\lambda, f) \to (F, e, d)$, where f is a boolean circuit, F is a garbled circuit, e is input-encoding information, and d is output-decoding information. Garble is a randomized algorithm, but the others are deterministic.
- $\mathsf{Encode}(e, x) \to X$, where x is a plaintext circuit input, and X is a corresponding garbled input.
- $\mathsf{Eval}(F, X) \to Y$, where Y is a garbled output.
- $\mathsf{Decode}(d, Y) \to y$, where y is a plaintext output.

We say that the garbling scheme is a **$\mathcal{G}$-scheme**, if it supports f that are $\mathcal{G}$-circuits.

Several properties are useful, and here we state the relevant security properties for the case of **gate-hiding** $\mathcal{G}$-schemes:

- **Correctness:** For all $(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f)$, we have

$$\mathsf{Decode}(d, \mathsf{Eval}(F, \mathsf{Encode}(e, x))) = f(x).$$

- **Gate-Hiding Privacy:** There exists a simulator $\mathcal{S}$, such that for all $\mathcal{G}$-circuits f and all inputs x, the following two distributions are indistinguishable:

$\underline{\mathrm{PrivReal}(1^\lambda, f, x)}$:
$(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f)$
$X := \mathsf{Encode}(e, x)$
return (F, X, d)

$\underline{\mathrm{PrivSim}^{\mathcal{S}}(1^\lambda, f, x)}$:
return $\mathcal{S}(1^\lambda, \mathsf{topo}(f), f(x))$

  In other words, (F, X, d) leaks no information beyond $\mathsf{topo}(f)$ and $f(x)$.

- **Gate-Hiding Obliviousness:** There exists a simulator $\mathcal{S}$, such that for all $\mathcal{G}$-circuits f and all inputs x, the following two distributions are indistinguishable:

$\underline{\mathrm{OblivReal}(1^\lambda, f, x)}$:
$(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f)$
$X := \mathsf{Encode}(e, x)$
return (F, X)

$\underline{\mathrm{OblivSim}^{\mathcal{S}}(1^\lambda, f, x)}$:
return $\mathcal{S}(1^\lambda, \mathsf{topo}(f))$

  In other words, (F, X) (without d) leaks no information beyond $\mathsf{topo}(f)$.

- **Authenticity:** For any $\mathcal{G}$-circuit f, input x, and efficient adversary $\mathcal{A}$, the following game outputs 1 with negligible probability:

$\underline{\text{Auth}^{\mathcal{A}}(1^\lambda, f, x)}$:
$(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f)$
$X := \mathsf{Encode}(e, x)$
$\tilde{Y} \leftarrow \mathcal{A}(F, X)$
if $\mathsf{Decode}(d, \tilde{Y}) \notin \{f(x), \perp\}$ then return 1 else return 0

2.3 Dual-key Hash

Our constructions require a function with type $H : \{0,1\}^* \times \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\ell$. To define the required notion of security, we introduce a related function:

$$O_K(t, a, X) = \begin{cases} H(t; X, K) & \text{if } a = 0 \\ H(t; K, X) & \text{if } a = 1 \end{cases}$$

We say that H is a **dual-key hash** if, for random choice of $K \leftarrow \{0,1\}^\lambda$, oracle access to O_K is indistinguishable from oracle access to a random function, against distinguishers who query O_K with distinct t-values.

Intuitively, one can think of H as a PRF with two keys. The outputs of H look random as long as one of the two keys is random and secret (the other key can be chosen by the adversary). This notion is similar to the "dual PRF" definition in [BL15], however the additional and non-repeating t-input in our setting makes realizing our notion easier.

One can instantiate H as a random oracle. In practice, one might simply use $H = \mathsf{sha256}$. In the standard model, one can use $H(t; A, B) = F(A, t) \oplus F(B, t)$, where F is a secure PRF. The fastest garbling schemes in practice use *fixed-key AES* as an ideal permutation, following [BHKR13], to take best advantage of AES hardware support. We think it likely that the schemes in this work can be adapted naturally to this setting. But since our focus is in part to minimize the hardness assumption of the schemes, we focus on the dual-key hash abstraction which can be instantiated from a plain PRF.

2.4 Basics of Garbled Circuit Techniques

We review several basic and standard techniques for garbled circuits. Readers familiar with the internals of recent garbled circuit constructions can skip this section.

Textbook Yao. Suppose a gate has input wire labels (A_0, A_1) and (B_0, B_1), and output wire labels (C_0, C_1). Here the subscripts correspond to the *truth value* (so A_0 is the wire label encoding false on that wire). Suppose we wish to garble an AND gate as an example, then we generate the following encryptions:

$$G_1 = \mathsf{Enc}(A_0, \mathsf{Enc}(B_0, C_0)) \qquad G_3 = \mathsf{Enc}(A_1, \mathsf{Enc}(B_0, C_0))$$
$$G_2 = \mathsf{Enc}(A_0, \mathsf{Enc}(B_1, C_0)) \qquad G_4 = \mathsf{Enc}(A_1, \mathsf{Enc}(B_1, C_1))$$

However, position in this list clearly leaks the truth value on each wire. So the list is randomly permuted. The evaluator, who receives just a single garbled input combination A_a, B_b is expected to perform trial decryption of each ciphertext. Hence, the encryption scheme must give some indication of whether decryption is successful.

Point-permute. Beaver, Micali, and Rogaway [BMR90] introduced a point-and-permute technique that is now used in essentially every practical garbling scheme. The idea is to append to each wire label a **color bit**. The two wire labels on each wire will have opposite color bits. The association between color bits and truth values is random and known only to the garbler. The evaluator, who sees only one label per wire, sees only a random color bit that is distributed independently of the truth value it represents.

As before, let a gate have input wire labels (A_0, A_1) and (B_0, B_1), and output wire labels (C_0, C_1). Unlike before, let the subscript denote the *color bit* of the wire label. The evaluator holds one label from each wire, and is allowed to use their (public) color bits to decide how to proceed. Suppose for example that A_0, B_1, and C_1 correspond to true on their respective wires. Then the garbled gate consists of four ciphertexts:

$$G_1 = \mathsf{Enc}(A_0, \mathsf{Enc}(B_0, C_0)) \qquad G_3 = \mathsf{Enc}(A_1, \mathsf{Enc}(B_0, C_0))$$
$$G_2 = \mathsf{Enc}(A_0, \mathsf{Enc}(B_1, C_1)) \qquad G_4 = \mathsf{Enc}(A_1, \mathsf{Enc}(B_1, C_0))$$

The ciphertexts can be arranged in precisely this order, since the order depends only on the (public) color bits and not the (secret) truth values. The evaluator can use the color bits to identify exactly which ciphertext to decrypt. There is no need for the evaluator to perform trial decryption on each ciphertext, and therefore no need to detect "correct decryption." This allows the scheme to use a simple encryption, namely:

$$G_1 = H(A_0, B_0) \oplus C_0 \qquad G_3 = H(A_1, B_0) \oplus C_0$$
$$G_2 = H(A_0, B_1) \oplus C_1 \qquad G_4 = H(A_1, B_1) \oplus C_0$$

Here H is a dual-key hash, defined in Section 2.3 (a unique nonce should also be given as input to each invocation of H, but we have omitted it from the notation).

Simple garbled row reduction. Naor, Pinkas, and Sumner [NPS99] introduced a method to reduce the size of garbled gates from 4 to 3 ciphertexts, called *garbled row reduction (GRR)*. The idea is to exploit the freedom in choosing the output wire labels C_0, C_1, which are not yet fixed at the time this gate is garbled. In particular, we can always make the first ciphertext G_1 equal to 0^λ. In the example above, we do so by choosing $C_0 = H(A_0, B_0)$. Since G_1 is always 0^λ, it does not need to be included in the garbled gate — the evaluator can "imagine" $G_1 = 0^\lambda$ and proceed as above.

3 Extending the KKS and WM Schemes

Kempka et al. [KKS16], and independently Wang & Malluhi [WM17], give constructions of a gate-hiding garbling scheme for the class of **symmetric gates.** We now review their schemes and discuss in more detail their limitation to the class of symmetric gates.

3.1 Overview of the Constructions

In a symmetric gate, we have $g(0,1) = g(1,0)$. The main idea in both KKS and WM is for the false wire label A_0 and true wire label A_1 on a wire to satisfy the relationship $A_1 = A_0 + \Delta \pmod{2^\lambda}$, where Δ is a global secret constant common to all wires. Suppose the evaluator has a wire label A_a and B_b, corresponding to input combination (a, b) on some gate. Adding these wire labels mod 2^λ results in one of $\{A_0 + B_0, A_0 + B_0 + \Delta, A_0 + B_0 + 2\Delta\}$. Importantly, adding the wire labels "collapses" the two input combinations $(0,1)$ and $(1,0)$ to the same value.

Both the KKS and WM construction use this idea. That is, the evaluator's first step is to add the input wire labels and use the result as input to a cryptographic hash, to obtain one of $\{H(A_0 + B_0), H(A_0 + B_0 +$

$\Delta), H(A_0 + B_0 + 2\Delta)\}$. The corresponding result is used as a key that allow the receiver to learn the correct output wire label. The two schemes diverge significantly in their techniques at this point (specifically, WM uses polynomial interpolations), but the most important idea is this method for encoding truth values as wire labels with a global correlation by Δ.

In WM, each garbled gate requires $2\lambda + 2$ bits. In the general case of KKS, each garbled gate requires $2\lambda + 8$ bits (but see the note below about optimizations for some special cases).

Hardness assumptions. Because of the way truth values are encoded into wire labels, these schemes require a non-standard assumption. To understand why, consider that an evaluator learns one of the keys $\{H(A_0+B_0), H(A_0+B_0+\Delta), H(A_0+B_0+2\Delta)\}$. The security proof will have to argue that the other two keys look random. Since for all gates, the keys are related by a common secret Δ, a related-key-type assumption is used. Furthermore, since these keys are used to encrypt other wire labels in the system, which are also related by the same Δ, a circularity assumption is necessary as well. We point the reader to the KKS/WM papers for more details, and to [CKKZ12] who describe the analogous situation that occurs when using Free-XOR.

Optimizations for input gates. The authors of KKS show further how to garble gates at the *input level* of the circuit for just $\lambda + 8$ bits, exploiting the extra freedom available for choosing input wire labels. In this work we focus only on the general case of *internal* gates of the circuit, whose input labels will be already fixed by the time the gate is being garbled. We justify this choice with the observation that the number of input wires is typically an extremely small fraction of the total wires in a circuit (*e.g.*, the SHA-256 circuit has 256 inputs but over 130,000 gates [GLNP15]), so these optimizations have a relatively small effect. Certainly the difference between internal gates costing 3λ vs. 2λ bits is significantly more important.

3.2 The Limitation to Symmetric Gates, and How to Overcome It

As mentioned in Section 1.1, the KKS and WM schemes can garble only **symmetric gates** — those g where $g(0,1) = g(1,0)$. In contrast, our new constructions can garble any gate except for the two constant gates $(x,y) \mapsto 0$ and $(x,y) \mapsto 1$.

Is there a trivial modification to these schemes that avoids this limitation? We argue that the limitation to $\mathcal{G}_{\mathsf{symm}}$ is *inherent* to these schemes, and cannot be avoided for free.

Consider an asymmetric gate $g(a,b) = a \wedge \overline{b}$ above. In order to express this gate in terms of a symmetric gate, one needs to incorporate the logic of a NOT-gate somehow. But looking closely at the choice of wire labels in KKS/WM, it seems that a NOT-gate can never be free. In particular, the wire labels satisfy $A_1 = A_0 + \Delta \pmod{2^\lambda}$ for a global Δ, and this relationship between wire labels is **not symmetric!** Contrast this with other garbling paradigms:

- In free-XOR [KS08] and its derivatives, the wire labels satisfy $A_1 = A_0 \oplus \Delta$, where $\oplus$ denotes bitwise XOR. This relation *is* symmetric, so $A_0 = A_1 \oplus \Delta$ as well.
- In textbook Yao, the GRR3 scheme of [NPS99], and the GRR2 scheme of [PSSW09], wire labels are unconstrained. A vacuous relation between wire labels is obviously symmetric.

In both of these cases, one can implement a NOT gate for free (obliviously) by simply having the garbler change which label he/she considers as the false one.[5]

[5] This fact is explicitly mentioned in [GLNP15].

With wire labels in the KKS/WM paradigm, however, the two wire labels simply have their truth values "baked-in", in a fundamental way. Their truth values cannot be swapped simply by the garbler changing his/her internal perspective.

The garbler could consider $(-\Delta \bmod 2^\lambda)$ to be the *local* wire-label-difference, just for this gate. But this would invert the truth value on *both* input wires, leaving the resulting gate functionality still symmetric. To realize a non-symmetric gates it must be necessary to negate only one of the two input wires.

Supporting non-symmetric gates via unary gates. We observe that every gate $g \in \mathcal{G}_{\mathsf{all}}$ can be expressed as $g(a,b) = f(h(a),b)$ where f is symmetric and h is a unary gate. For example, a non-symmetric gate like $g(a,b) = a \wedge \overline{b}$ can be written using DeMorgan's laws as $g(a,b) = \textsc{nor}(\textsc{not}(a),b)$. Hence, an obvious way to extend KKS/WM beyond $\mathcal{G}_{\mathsf{symm}}$-gates is to incorporate unary gates.

Including unary gates in KKS/WM We show how to incorporate unary gates into KKS/WM, in a way that hides the choice of unary gate and costs λ bits per gate. The idea is to start with the textbook Yao construction and then apply a row-reduction in the style of [NPS99]. Let A_0 and $A_1 = A_0 + \Delta$ be the input wire labels, and let B_0 and $B_1 = B_0 + \Delta$ be the output wire labels (yet to be determined). In the textbook Yao scheme, the garbled gate will include two ciphertexts:

$$H(A_0) \oplus B; \qquad H(A_1) \oplus B'$$

where $B, B' \in \{B_0, B_1\}$, depending on which unary gate we are garbling. These two ciphertexts will be permuted according to the color bits (see Section 2.4) of A_0/A_1.

Suppose after permuting, the ciphertext $H(A_0) \oplus B$ is first. Then the row-reduction idea is to choose $B = H(A_0)$ so that this ciphertext is always all zeroes. Having fixed one of the output wire labels B in this way, the garbler can solve for the other wire label $\{B_0, B_1\} \setminus B$ so that the labels satisfy the relation $B_1 = B_0 + \Delta$.

Since the first of the two ciphertexts is always all zeroes, it does not need to be sent. The evaluator will simply hash its input wire label in the case it has color bit 0; otherwise it will hash its wire label and XOR with the ciphertext. So the garbled gate consists of a single λ-bit ciphertext.

Now, one can garble every $\mathcal{G}_{\mathsf{all}}$-gate using a gadget combining a unary gate and symmetric gate, in the manner just described. Standard techniques can be used to show that this unary gate construction is secure and hides the choice of unary gate. Hence:

Claim 1. *Using the above modification of the KKS (resp. WM) constructions, a $\mathcal{G}_{all}$-circuit can be garbled in a gate-hiding way, with cost $3\lambda + 8$ (resp. $3\lambda + 2$) per gate.*

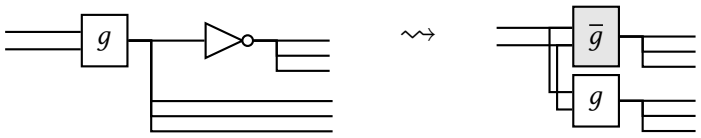
Dealing with $\mathcal{G}_{\mathsf{all}}$-gates in KKS/WM. Suppose we want to garble a circuit expressed in terms of $\mathcal{G}_{\mathsf{all}}$ (or even $\mathcal{G}_{\mathsf{non\text{-}const}}$) gates. When using the KKS/WM paradigm, we see the following possibilities, all of which involve extra costs (either in size of the garbled circuit or in leaking more information):

1. Leak the distinction between symmetric & non-symmetric gates (but leak nothing else about the gate type), resulting in $2\lambda + O(1)$ bits for symmetric and $3\lambda + O(1)$ bits for non-symmetric. In some cases, like the semi-private function evaluation protocol of [PSS09], all parties know that certain gates will be non-symmetric (pass-through gates / multiplexers with the selection bit fixed).[6]

[6] When discussing this scenario, Kempka *et al.* acknowledge that their scheme cannot readily garble a multiplexer. They suggest to use a traditional GRR3 garbling scheme for these gates. Both their suggestion and ours in this section would require $3\lambda + O(1)$ bits per non-symmetric gate.

2. Hide the distinction between symmetric & non-symmetric gates by garbling every gate as a gadget of the form $f(h(a), b)$, where f is symmetric and h is unary. Such a gadget is necessary since non-symmetric gates require it for functionality, and we want all gates to use the same garbling technique. This results in a cost of $3\lambda + O(1)$ bigs for all gates.

3. In the case of a NOT gate, we can "absorb" it into the upstream gate that feeds into it. However, because the upstream gate may have high fan-out, this requires duplicating the upstream gate. If the upstream gate $g(a, b)$ is symmetric, then the clone $\neg g(a, b)$ is also symmetric. This increases the number of gates in the circuit and leaves noticeable artifacts in the circuit topology (even if the gate types are hidden, it is evident from the topology that the two gates take the same inputs). Whether these artifacts are problematic depends on the specific application of garbled circuits.



4 Construction based on Polynomial Interpolation

We review a garbling scheme of Pinkas et al. [PSSW09] (hereafter PSSW), that results in garbled gates of $2\lambda + 4$ bits. The scheme uses different garbling/evaluation approaches depending on whether the gate has even/odd parity (the parity of a gate is even if its truth table contains an even number of 1s). We then show that a simple modification allows the scheme to be gate-hiding for all $\mathcal{G}_{\text{non-const}}$-gates (i.e., gates of either parity).

4.1 Overview of PSSW Garbling Scheme

The PSSW approach is based on polynomial interpolation. We describe only the method they propose for odd-parity gates, since that is the method we will extend to the even-parity case as well.

Suppose a gate has input wire labels (A_0, A_1) and (B_0, B_1). Here, the subscripts will denote the **color bits** (see Section 2.4) of the wires. This means that the evaluator will be able to behave differently depending on the subscripts of the input wire labels he/she has.

For each input combination, we define an associated "key" value:

$$K_1 = H(A_0, B_0); \qquad K_3 = H(A_1, B_0);$$
$$K_2 = H(A_0, B_1); \qquad K_4 = H(A_1, B_1).$$

The evaluator learns only one combination of input wire labels, and hence learns one of the K_i values (and knows the subscript of this K_i value, as it is determined by the input wires' color bits). The property we require of H is that the other three K_i values look random.

For an AND gate, we want to arrange things so that learning K_i allows the evaluator to learn the corresponding output wire label of the gate. For instance, depending on the color bits, we might need to arrange for the evaluator to learn the following:

knowing K_1 lets you learn C_0	knowing K_3 lets you learn C_0
knowing K_2 lets you learn C_1	knowing K_4 lets you learn C_0

For this example, the garbler can proceed as follows. First, use polynomial interpolation to find the unique degree-2 polynomial P passing through the points $\{(1, K_1), (3, K_3), (4, K_4)\}$. These are the cases for which

the evaluator should learn C_0. Here we take the K_i values to live in $\mathbb{Z}_p$ for some prime p, and P is a polynomial over $\mathbb{Z}_p$. In practice, one would instead use a finite field $GF(2^\lambda)$, but a prime field makes the notation a little simpler.

Now the garbler finds the unique degree-2 polynomial Q passing through the points $\{(2,K_2),(5,P(5)),(6,P(6))\}$. Hence, we have:

- interpolating a polynomial through $(1,K_1),(5,P(5)),(6,P(6))$ results in P
- interpolating a polynomial through $(2,K_2),(5,P(5)),(6,P(6))$ results in Q
- interpolating a polynomial through $(3,K_3),(5,P(5)),(6,P(6))$ results in P
- interpolating a polynomial through $(4,K_4),(5,P(5)),(6,P(6))$ results in P

This suggests to make the garbled gate consist of values $P(5),P(6)$ and to set $P(0)$ and $Q(0)$ to be the false/true output wire labels. The evaluator will compute K_i, interpolate a polynomial using the garbled gate values, and then evaluate that polynomial at point 0.

Dealing with color bits. Recall that each wire label is associated with an additional color bit. The garbled gate must also contain information that tells the evaluator the color bit of the output wire label.

To do so, we interpret H as a function of the form $H : \{0,1\}^* \to \{0,1\}^{\lambda+1}$, and write $K_i \| \kappa_i = H(A,B)$, where $K_i \in \{0,1\}^\lambda$ and $\kappa_i \in \{0,1\}$. The garbler can choose random color bits for the output wire labels. Then for each i, he/she can encrypt the color bit of the appropriate output wire label, using κ_i as a one-time pad. This adds an additional 4 bits to the garbled gate. The evaluator will choose which 1-bit ciphertext to decrypt based on the color bits of the input wires.

4.2 Modification to Support Both Even/Odd Gates

Our main contribution in this section is to simply point out that the PSSW technique for odd gates can be extended to work for even gates as well.

We keep the same evaluation algorithm as in PSSW. Recall that the evaluator interpolates a polynomial through points $P(5)$ and $P(6)$ (which comprise the garbled gate information) and evaluates that polynomial at point 0 to obtain the output wire label. In order for this same evaluation procedure to work for an even gate, it suffices to arrange the polynomials so that **2 of the points are on P and 2 are on Q.** As an example, suppose we want K_1,K_2 to lie on P and K_3,K_4 to lie on Q. Then we require polynomials that satisfy:

$$\left\{\begin{array}{rl} P(1) & = K_1 \\ P(2) & = K_2 \\ Q(3) & = K_3 \\ Q(4) & = K_4 \\ P(5)-Q(5) & = 0 \\ P(6)-Q(6) & = 0 \end{array}\right\} \iff \begin{bmatrix} 1^0 & 1^1 & 1^2 & & & \\ 2^0 & 2^1 & 2^2 & & & \\ & & & 3^0 & 3^1 & 3^2 \\ & & & 4^0 & 4^1 & 4^2 \\ 5^0 & 5^1 & 5^2 & -5^0 & -5^1 & -5^2 \\ 6^0 & 6^1 & 6^2 & -6^0 & -6^1 & -6^2 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ q_0 \\ q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \\ K_4 \\ 0 \\ 0 \end{bmatrix}$$

where we write $P(x) = p_0 + p_1 x + p_2 x^2$ and $Q(x) = q_0 + q_1 x + q_2 x^2$.

Note that the K_i values are fixed (the generator has no control over them, since the input wire labels are already fixed). Hence, the generator will compute the K_i values and solve for polynomials P and Q using the above linear equation. Clearly it suffices for the 6×6 matrix to be invertible.

In the general case, the garbler will consider the following 6×6 matrix:

$$\begin{bmatrix} \tau_{00}1^0 & \tau_{00}1^1 & \tau_{00}1^2 & \overline{\tau_{00}}1^0 & \overline{\tau_{00}}1^1 & \overline{\tau_{00}}1^2 \\ \tau_{01}2^0 & \tau_{01}2^1 & \tau_{01}2^2 & \overline{\tau_{01}}2^0 & \overline{\tau_{01}}2^1 & \overline{\tau_{01}}2^2 \\ \tau_{10}3^0 & \tau_{10}3^1 & \tau_{10}3^2 & \overline{\tau_{10}}3^0 & \overline{\tau_{10}}3^1 & \overline{\tau_{10}}3^2 \\ \tau_{11}4^0 & \tau_{11}4^1 & \tau_{11}4^2 & \overline{\tau_{11}}4^0 & \overline{\tau_{11}}4^1 & \overline{\tau_{11}}4^2 \\ 5^0 & 5^1 & 5^2 & -5^0 & -5^1 & -5^2 \\ 6^0 & 6^1 & 6^2 & -6^0 & -6^1 & -6^2 \end{bmatrix} \tag{1}$$

for some $(\tau_{00}, \tau_{01}, \tau_{10}, \tau_{11}) \in \{0,1\}^4 \setminus \{0000, 1111\}$ (corresponding to a non-constant gate). One can verify by hand that the above matrix is invertible in all cases. This is also true in $GF(2^\lambda)$ if we replace integers $\{1, \ldots, 6\}$ with finite field elements whose representations in hex are $\{\texttt{0x1}, \ldots, \texttt{0x6}\}$.

4.3 Details

In Figure 2 we give a generic template for a garbling scheme. Both of our new constructions will instantiate this template. In the template for the garbler, we use W_i^b as the wire label representing truth value b on wire i, and σ_i to denote the color bit on W_i^0. For the evaluator, we use W_i^* as the "active" wire label (representing unknown truth value) on wire i, and χ_i to denote its color bit (known to the evaluator).

We give the specific procedure for garbling & evaluating gates of this scheme in Figure 3. Departing from the (simplified) discussion above, we include the color bits in the input to H to ensure that all "nonce" arguments to H are globally unique (as required by the dual-key hash definition); e.g., $K_{ab} \| \kappa_{ab} := H(\mathsf{idx}, \underline{a}, \underline{b}; \cdots)$.

The scheme is written to involve an explicit matrix inverse, as in Equation 1 above. In practice, there are only $16 - 2 = 14$ possible matrices (one for each non-constant setting of the τ_{ab} bits), and their inverses would all be easily hard-coded into a lookup table.

4.4 Security

Theorem 2. *The construction in Figures 2+3 satisfies the gate-hiding privacy, obliviousness (both with respect to $\mathcal{G}_{\text{non-const}}$), and authenticity properties (Section 2.2), if H is a dual-key hash.*

Proof. We start with the proof of the privacy property. The proof uses the following sequence of hybrids.

Hybrid $(0,1)$**:** corresponds to the real interaction. The game obtains f, x, and runs $(F, e, d) \leftarrow \mathsf{Garble}(1^\lambda, f)$; $X := \mathsf{Encode}(e, x)$ and returns (F, X, d).

Hybrid $(0,2)$**:** Let v_i denote the truth value on wire i when the input to the circuit is x. Instead of choosing random σ_i for each wire, we choose random $\chi_i \leftarrow \{0,1\}$ and set $\sigma_i = \chi_i \oplus v_i$. Although this has no effect on the adversary's view, it helps to put the garbling process in the "frame of reference" of the evaluator (*i.e.*, in terms of the "active" wire label's color bit χ_i) instead of the garbler's (in terms of the false label's color bit σ_i).

We next proceed in a sequence of hybrids, several for each gate $i \in \mathsf{wires}(f)$. We assume the wires of the circuit $\mathsf{gates}(f)$ are ordered topologically.

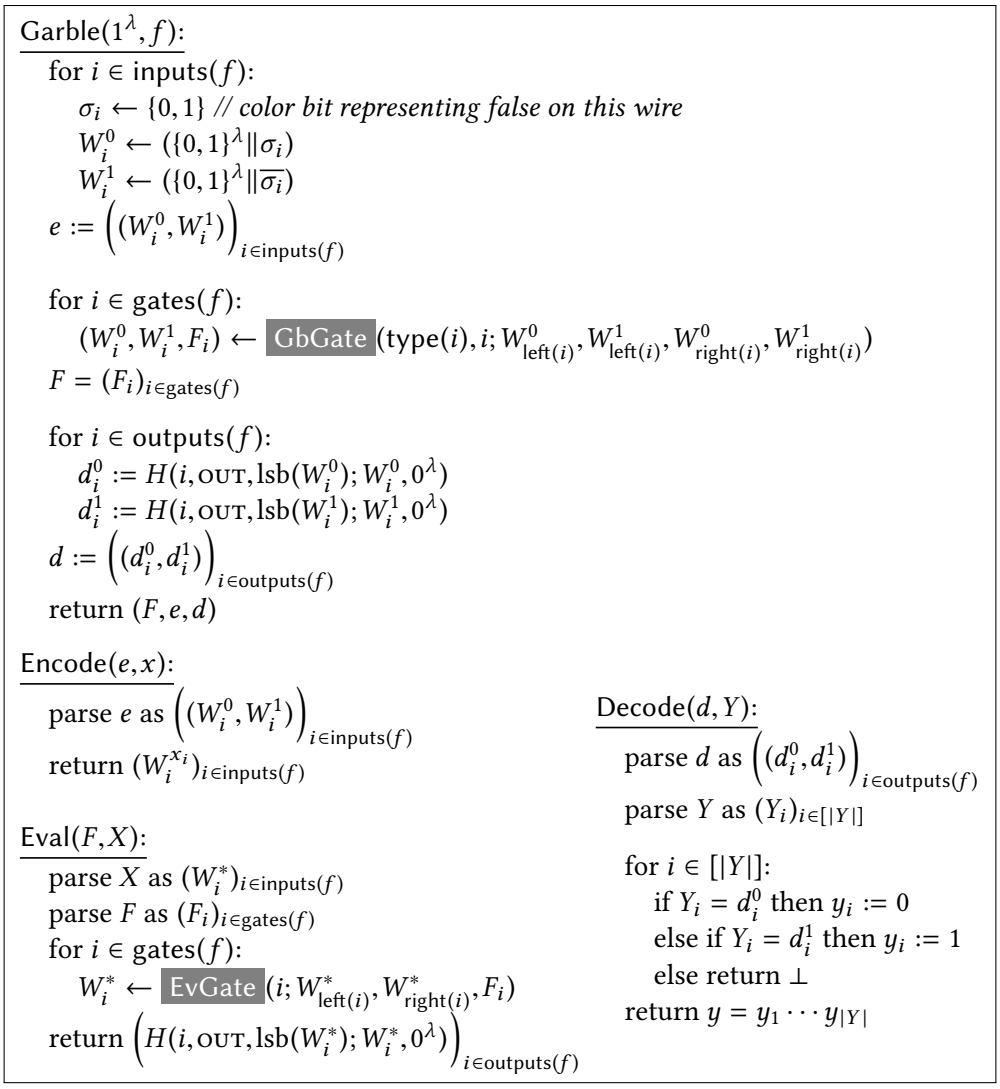


Figure 2: Generic template for garbling scheme. GbGate and EvGate subroutines to be specified later. Notation type(i) (used in Garble) refers to the gate functionality of the gate with index i (see Section 2.1).

Hybrid $(i, 0)$**:** Rename $\{W_i^*, \overline{W}_i^*\} = \{W_i^0, W_i^1\}$, where W_i^* is the label whose color bit is χ_i (*i.e.*, W_i^* is the "active" label that we expect the evaluator to hold for this wire).

Our invariant will be that in the hybrid $(i-1, 2)$, label $\overline{W}_i^*$ is chosen uniformly at random (this is certainly true for input wires by construction, and will be established inductively for internal wires). In this hybrid, instead of choosing $\overline{W}_i^*$ explicitly, we play as an adversary in the dual-key security game, and implicitly set $\overline{W}_i^*$ to be the value K chosen by the dual-key oracle O_K. The distribution will be the same, it merely suffices to show we can simulate all the uses of $\overline{W}_i^*$ while acting as adversary in the dual-key experiment.

If $i \in$ inputs(f), then by construction only W_i^* (not $\overline{W}_i^*$) is given explicitly to the distinguisher as part of garbled input X. The only other place $\overline{W}_i^*$ might be used is in a call to H, where the simulation knows the other arguments to H. These outputs of H can be computed as a dual-key adversary.

// INPUT: gate with functionality g and index idx;
left wire labels (A_0, A_1); right wire labels (B_0, B_1)
// OUTPUT: output wire labels (C_0, C_1); garbled gate information

$\underline{\mathsf{GbGate}(g, \mathsf{idx}; A_0, A_1, B_0, B_1)}$:

$\sigma_A := \mathrm{lsb}(A_0)$
$\sigma_B := \mathrm{lsb}(B_0)$
$\sigma_C \leftarrow \{0,1\}$

for $a, b \in \{0,1\}^2$: *// each combination of visible color bits:*
$\tau_{ab} := g(\sigma_A \oplus a, \sigma_B \oplus b)$ *// gate output truth value*
$K_{ab} \| \kappa_{ab} := H(\mathsf{idx}, a, b; A_{\sigma_A \oplus a}, B_{\sigma_B \oplus b})$

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ q_0 \\ q_1 \\ q_2 \end{bmatrix} := \begin{bmatrix} \tau_{00} & \tau_{00}\,\mathtt{0x1} & \tau_{00}\,\mathtt{0x1}^2 & \overline{\tau_{00}} & \overline{\tau_{00}}\,\mathtt{0x1} & \overline{\tau_{00}}\,\mathtt{0x1}^2 \\ \tau_{01} & \tau_{01}\,\mathtt{0x2} & \tau_{01}\,\mathtt{0x2}^2 & \overline{\tau_{01}} & \overline{\tau_{01}}\,\mathtt{0x2} & \overline{\tau_{01}}\,\mathtt{0x2}^2 \\ \tau_{10} & \tau_{10}\,\mathtt{0x3} & \tau_{10}\,\mathtt{0x3}^2 & \overline{\tau_{10}} & \overline{\tau_{10}}\,\mathtt{0x3} & \overline{\tau_{10}}\,\mathtt{0x3}^2 \\ \tau_{11} & \tau_{11}\,\mathtt{0x4} & \tau_{11}\,\mathtt{0x4}^2 & \overline{\tau_{11}} & \overline{\tau_{11}}\,\mathtt{0x4} & \overline{\tau_{11}}\,\mathtt{0x4}^2 \\ 1 & \mathtt{0x5} & \mathtt{0x5}^2 & 1 & \mathtt{0x5} & \mathtt{0x5}^2 \\ 1 & \mathtt{0x6} & \mathtt{0x6}^2 & 1 & \mathtt{0x6} & \mathtt{0x6}^2 \end{bmatrix}^{-1} \begin{bmatrix} K_{00} \\ K_{01} \\ K_{10} \\ K_{11} \\ 0 \\ 0 \end{bmatrix} \text{// } GF(2^\lambda)$$

$C_0 := p_0 \| \sigma_C$
$C_1 := q_0 \| \overline{\sigma_C}$

$G := p_0 + p_1 \cdot \mathtt{0x5} + p_2 \cdot \mathtt{0x5}^2$ *// $GF(2^\lambda)$*
$G' := p_0 + p_1 \cdot \mathtt{0x6} + p_2 \cdot \mathtt{0x6}^2$ *// $GF(2^\lambda)$*

for $a, b \in \{0,1\}^2$:
set $c_{ab} := \kappa_{ab} \oplus \sigma_C \oplus \tau_{ab}$ *// encrypted output wire label color bit*

return $(C_0, C_1, (G, G', c_{00}, c_{01}, c_{10}, c_{11}))$

// INPUT: gate index idx; left wire label A^*; right wire label B^*; garbled gate info
// OUTPUT: output wire label C^*

$\underline{\mathsf{EvGate}(\mathsf{idx}; A^*, B^*, (G, G', c_{00}, c_{01}, c_{10}, c_{11}))}$:

$\chi_A := \mathrm{lsb}(A^*)$
$\chi_B := \mathrm{lsb}(B^*)$
$K^* \| \kappa^* := H(\mathsf{idx}, \chi_A, \chi_B; A^*, B^*)$
$R :=$ unique degree-2 polynomial in $GF(2^\lambda)$ passing through
$\{(2\chi_A + \chi_B + 1, K^*), (5, G), (6, G')\}$
$\chi_C := c_{\chi_A, \chi_B} \oplus \kappa^*$
return $C^* = R(0) \| \chi_C$

Figure 3: Our gate-hiding garbling scheme based on polynomial evaluation.

Hybrid $(i, 1)$**:** We replace the dual-key oracle O_K with a random function. The change is indistinguishable from the previous hybrid by the dual-key security property. As a result, every call to H that involved $\overline{W}_i^*$ is replaced with a uniformly chosen value.

Hybrid $(i,2)$**:** For every gate $j > i$ such that $\max\{\mathsf{left}(j), \mathsf{right}(j)\} = i$, we make the following change in the call to $\mathrm{GbGate}(\cdot, j; \cdots)$:

Within this call to GbGate, three of the $K_{ab} \| \kappa_{ab}$ values are now being chosen uniformly at random. By construction, the one that is not being chosen uniformly at random corresponds to $(a,b) = (\chi_{\mathsf{left}(i)}, \chi_{\mathsf{right}(i)})$.

Observe that (p_0, q_0, G, G') are all linear combinations of $(K_{00}, \ldots, K_{11})$. In particular,

$$\begin{bmatrix} p_0 \\ q_0 \\ G \\ G' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & \texttt{0x5} & \texttt{0x5}^2 & 0 & 0 & 0 \\ 1 & \texttt{0x6} & \texttt{0x6}^2 & 0 & 0 & 0 \end{bmatrix} \times M^{-1} \times \begin{bmatrix} K_{00} \\ K_{01} \\ K_{10} \\ K_{11} \\ 0 \\ 0 \end{bmatrix}$$

where M is the matrix shown in Figure 3. Since M is invertible, the terms on the left-hand-side are *linearly independent* linear combinations of $(K_{00}, \ldots, K_{11})$.

So instead of fixing the 4 K_{ab} values (setting 3 of them uniformly at random) and solving for the left-hand side, we instead choose randomly G, G', and whichever of $\{p_0, q_0\}$ is associated with $\overline{W}_j^*$. Then we solve for the remaining value (W_j^*) by doing whatever the evaluator would do with G, G' and $K_{\chi_{\mathsf{left}(i)}, \chi_{\mathsf{right}(i)}}$.

Similarly, note that 3 of the 4 c_{ab} values are uniformly random because the corresponding one-time pad mask κ_{ab} is uniform. The other c_{ab} is uniform because it is a one-time pad encryption of χ_j, which is also being chosen uniformly. It makes no change in the distributions to generate all the c_{ab} values uniformly at random, and then set $\chi_j := c_{\chi_{\mathsf{left}(i)}, \chi_{\mathsf{right}(i)}} \oplus \kappa_{\chi_{\mathsf{left}(i)}, \chi_{\mathsf{right}(i)}}$.

After these changes, the distribution over all values is the same as before, but now the entire garbled gate $(G, G', c_{00}, \ldots, c_{11})$ and $\overline{W}_j^*$ are *explicitly* being chosen uniformly at random. This maintains the invariant of "inactive" wire labels being chosen uniformly.

Wrapping it up. We have argued that the hybrids are indistinguishable in the following sequence:

$$\begin{aligned} (0,1) \equiv (0,2) \equiv (1,0) &\approx (1,1) \equiv (1,2) \\ &\equiv (2,0) \approx (2,1) \equiv (2,2) \\ &\vdots \\ &\equiv (n,0) \approx (n,1) \equiv (n,2) \end{aligned}$$

where n is the number of wires in the circuit.

Let us summarize what is happening in the final hybrid. The simulator first chooses random wire labels for all of the garbled input X. It then chooses the entire garbled circuit to be uniformly random. Note that the truth values v_i on the wires are not needed for any of this, nor are the types of the gates. The only place v_i values are being used is to compute d: it is implicitly being used to determine which of d_i^0, d_i^1 is the uniform value and which is the one obtained by calling H. But for an output wire $i \in \mathrm{outputs}(f)$, the value v_i on the wire is just a bit of the circuit output.

This shows that the simulator in the final hybrid requires only $f(x)$ and $\mathrm{topo}(f)$ to operate. This completes the proof of the privacy property.

Obliviousness & Authenticity The proofs of obliviousness and authenticity follow easily from the proof of privacy.

- In the last hybrid in the proof of privacy, the only place the circuit output is used by the simulator is in the computation of decoding information d. In the obliviousness security experiment, the simulator

does not need to provide d. The simulation of (F, X) can be done without knowing circuit output $f(x)$. This shows that the scheme satisfies the **obliviousness** property.

- The security game that defines **authenticity** only requires (F, X, d) values. Hence, we can apply the same sequence of hybrids to that security game. In the last hybrid, the "inactive" garbled outputs $d_i^{1-v_i}$ are chosen uniformly at random. It is therefore with negligible probability that the adversary will guess one of them and succeed in the game. □

5 Construction that avoids Polynomial Interpolation

5.1 Overview of GLNP Garbling Scheme

In [GLNP15], Gueron et al. describe a different way to garble gates (in their case, odd-parity gates only) at a cost of 2 ciphertexts. While the construction of [PSSW09] involves polynomial interpolation, the construction of [GLNP15] involves only cheap XOR operations, making it preferable both in performance and ease of implementation.

The main idea is to start with the classical point-and-permute Yao scheme, in which the garbled gate consists of 4 ciphertexts. Suppose a gate has input wire labels (A_0, A_1) and (B_0, B_1), and output wire labels (C_0, C_1). As before, the subscripts correspond to the visible color bits. Consider the following example odd-parity gate in the textbook Yao scheme:

$$G_1 = H(A_0, B_0) \oplus C_0 \qquad G_3 = H(A_1, B_0) \oplus C_0$$
$$G_2 = H(A_0, B_1) \oplus C_1 \qquad G_4 = H(A_1, B_1) \oplus C_0$$

The high level idea is to exploit the two degrees of freedom in the choice of output labels C_0 and C_1, which are not yet chosen at the time this gate will be garbled. The first step is to apply the GRR3 row reduction of [NPS99], setting $G_1 = 0^\lambda$. In this example, we can do so by choosing C_0 in a special way: $C_0 = H(A_0, B_0)$.

After fixing one of the output wire labels, the next step is to fix the other output label so that $G_2 \oplus G_3 \oplus G_4 = 0^\lambda$. In this example, we can do so by choosing $C_1 = H(A_0, B_1) \oplus H(A_1, B_0) \oplus H(A_1, B_1)$.

By choosing C_0 and C_1 in this way, the garbler has guaranteed that $G_1 = G_2 \oplus G_3 \oplus G_4 = 0^\lambda$. Hence, G_1 does not need to be sent (it is always all zeroes), and neither does G_4 (it can always be reconstructed by the receiver as $G_2 \oplus G_3$). In this way, only two ciphertexts actually need to be sent.

The problem of even-parity gates. One can check that the GLNP technique works for *any* odd-parity gate, but does **not** work when the gate has even parity. We illustrate with an example. Consider the following classical-Yao garbled gate for an even-parity truth table:

$$G_1 = H(A_0, B_0) \oplus C_0 \qquad G_3 = H(A_1, B_0) \oplus C_1$$
$$G_2 = H(A_0, B_1) \oplus C_1 \qquad G_4 = H(A_1, B_1) \oplus C_0$$

To achieve $G_1 = 0^\lambda$ we must set $C_0 = H(A_0, B_0)$ as before. But now observe that $G_2 \oplus G_3 \oplus G_4 = H(A_0, B_1) \oplus H(A_1, B_0) \oplus H(A_1, B_1) \oplus C_0$, a value that is already fixed! Because of the even parity of the gate, the C_1 terms cancel out in this expression. There is no way to choose C_1 so that $G_2 \oplus G_3 \oplus G_4 = 0^\lambda$ as before.

5.2 Our Construction

Let us look a little more abstractly at the GLNP scheme. The evaluator computes K (the hash of the two wire labels) and then obtains the final wire label as $K \oplus \alpha G \oplus \beta G'$, where G and G' are the two "ciphertexts"

in the garbled gate, and α and β are bits that depend on color bits of the wire labels. The mapping between color bits and α, β coefficients is *fixed* for the entire scheme.

Our approach is to *randomize* and *partially hide* this mapping of color bits to α, β coefficients. In our scheme, evaluation works as follows:

- The evaluator hashes the input wire labels to compute a key K.
- The evaluator uses K to decrypt a 2-bit ciphertext containing $\alpha \| \beta$. There are four such 2-bit ciphertexts, arranged according to color bits. The evaluator uses the color bits of the input labels to determine which of these 2-bit ciphertexts to decrypt.
- Having obtained the appropriate α, β, the evaluator computes the output wire label as $K \oplus \alpha G \oplus \beta G'$.

So each garbled gate consists of the following:

- G and G' (2λ bits)
- four 2-bit ciphertexts that encrypt α, β values (8 bits total)
- four 1-bit ciphertexts that encrypt the color bit of the output wire label (4 bits total)

The real power of the scheme comes from the *indirection* in conveying the evaluator's final linear combination. The evaluator uses his/her color bits to decrypt a constant-sized ciphertext, which tells him/her what linear combination to finally apply. A similar kind of indirection also appears in the construction of [KKS16], where they use it to circumvent a lower bound for "linear garbling" from [ZRE15] (the model for the lower bound implicitly assumes a direct, fixed correspondence between color bits and the evaluator's final linear combination).

Now let us consider how the garbler can arrange for all of this to happen. Let C_0 and C_1 denote the false/true output wire labels (yet to be determined). Let $K_1, \ldots, K_4$ be the four possible input hashes, as before. Let α_i, β_i denote the coefficients that the evaluator will use when he/she has K_i. Below is an example of the correctness conditions required for an example gate:

$$\begin{aligned} C_0 &= K_1 \oplus \alpha_1 G \oplus \beta_1 G' \\ C_0 &= K_2 \oplus \alpha_2 G \oplus \beta_2 G' \\ C_1 &= K_3 \oplus \alpha_3 G \oplus \beta_3 G' \\ C_0 &= K_4 \oplus \alpha_4 G \oplus \beta_4 G' \end{aligned} \quad \Longleftrightarrow \quad \begin{bmatrix} K_1 \\ K_2 \\ K_3 \\ K_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \alpha_1 & \beta_1 \\ 1 & 0 & \alpha_2 & \beta_2 \\ 0 & 1 & \alpha_3 & \beta_3 \\ 1 & 0 & \alpha_4 & \beta_4 \end{bmatrix} \begin{bmatrix} C_0 \\ C_1 \\ G \\ G' \end{bmatrix} \tag{2}$$

We let the garbler choose $\{\alpha_i, \beta_i\}$ values uniformly subject to the matrix in Equation 2 being invertible. Importantly, for different gate types, *this is a different distribution* over the α_i, β_i values! In particular, when the gate has odd parity (i.e., the parity of the first column is odd), there are 96 ways to choose α_i, β_i coefficients to make the matrix invertible. When the gate has even parity, there are 104 ways. Beyond that, the distributions are different even for different gates of the same parity. Below we discuss in more detail how this difference affects security.

In summary, the garbler computes $K_1, \ldots, K_4$ (these are fixed by the choice of the input wire labels), chooses random α_i, β_i values that make the appropriate matrix invertible, and finally solves for consistent C_0, C_1, G, G' according to Equation 2. The C_0, C_1 values will be the output wire labels and (G, G') will be the garbled gate (along with the 12 bits of encryptions mentioned above).

Why it hides the gate type. From the evaluator's perspective, every kind of gate is handled the same way — decrypt the correct α, β and output $K \oplus \alpha G \oplus \beta G'$.

However, the garbler's behavior depends on the choice of gate. In particular, he/she uses a different distribution over the α_i, β_i values for different gates. We must ensure that this difference is not noticeable to the evaluator. The key point is that the evaluator sees **only a single α_i, β_i pair** while the other coefficients remain encrypted.[7] Furthermore, all possible garbling distributions have the property that for every i, the marginal distribution of (α_i, β_i) is uniform. Hence, the evaluator sees only a uniform α_i, β_i, regardless of the gate type.

To see why this is true, take any vector $v \in \{0,1\}^4 \setminus \{0000, 1111\}$ and consider any invertible matrix (over $\mathbb{Z}_2$) of the following form:

$$\begin{bmatrix} v_1 & \overline{v_1} & \alpha_1 & \beta_1 \\ v_2 & \overline{v_2} & \alpha_2 & \beta_2 \\ v_3 & \overline{v_3} & \alpha_3 & \beta_3 \\ v_4 & \overline{v_4} & \alpha_4 & \beta_4 \end{bmatrix} \tag{3}$$

Note that flipping every bit in the 3rd column is an elementary matrix operation, since the first two columns sum to the all-ones vector. Hence this modification has no effect on the determinant. This modification is also invertible. Thus, for any i, we have an 1-to-1 correspondence between the set of invertible matrices with (α_i, β_i) and those with $(\overline{\alpha_i}, \beta_i)$. Of course, the same can be said for the mappings that flip every bit in the 4th column, or in both the 3rd and 4th columns. Hence, after fixing v, the number of ways to complete the matrix in an invertible way does not depend on the choice of a particular (α_i, β_i).

More details about evaluation. Similar to the previous construction, we assume a cryptographic hash function of the form $H : \{0,1\}^* \to \{0,1\}^{\lambda+3}$. When the evaluator has input wire labels A and B, he/she evaluates $K \| \kappa \| \widehat{\kappa} \leftarrow H(A, B)$, where $K \in \{0,1\}^\lambda$, $\kappa \in \{0,1\}$, and $\widehat{\kappa} \in \{0,1\}^2$. Then:

- $\widehat{\kappa}$ is used as a one-time pad key to decrypt the α, β coefficients.
- K is used to compute the output wire label as $K \oplus \alpha G \oplus \beta G'$.
- κ is used as a one-time pad key to decrypt the output wire label's color bit.

As in the previous scheme, some of the garbler's computations can be hard-coded into lookup tables in an implementation. Each possible value of $\tau_{00} \cdots \tau_{11}$ (14 choices) can index a list of valid α_{ab}, β_{ab} values along with the inverse of the appropriate matrix from Equation 3. These lookup tables will clearly be more extensive for this scheme than for the previous one, but overall have reasonable constant size.

5.3 Formal Details & Security

In Figure 4 we give the formal details of the construction. It follows the high-level discussion above.

Theorem 3. *The construction in Figures 2+4 satisfies the gate-hiding privacy, obliviousness (both with respect to $\mathcal{G}_{\text{non-const}}$), and authenticity properties (Section 2.2), if H is a dual-key hash.*

Proof. The proof follows the overall structure of the proof of Theorem 2. Hence, we concentrate on the differences.

As in the previous proof, we change perspective so that the garbling process is expressed in the evaluator's point of view. For each wire, W_i^* denotes the anticipated "active" wire label and $\overline{W}_i^*$ denotes the "inactive" one.

[7] If all the coefficients were to be made public, then we are back in the original situation of [GLNP15], and leaking the parity of the gate seems inevitable for this choice of evaluation equation.

// INPUT: gate with functionality g and index idx;
left wire labels (A_0, A_1); right wire labels (B_0, B_1)
// OUTPUT: output wire labels (C_0, C_1); garbled gate information

$\underline{\mathsf{GbGate}(g; A_0, A_1, B_0, B_1)}$:

$\sigma_A := \mathrm{lsb}(A_0)$
$\sigma_B := \mathrm{lsb}(B_0)$
$\sigma_C \leftarrow \{0,1\}$

for $a, b \in \{0,1\}^2$: *// each combination of visible color bits:*
$\tau_{ab} := g(\sigma_A \oplus a, \sigma_B \oplus b)$ *// gate output truth value*
$K_{ab} \| \kappa_{ab} \| \widehat{\kappa}_{ab} := H(\mathsf{idx}, a, b; A_{\sigma_A \oplus a}, B_{\sigma_B \oplus b})$

sample $\{\alpha_{ab}, \beta_{ab} \mid a, b \in \{0,1\}\}$ uniformly s.t. the matrix below is invertible

$$\begin{bmatrix} \tilde{C}_0 \\ \tilde{C}_1 \\ G \\ G' \end{bmatrix} := \begin{bmatrix} \overline{\tau_{00}} & \tau_{00} & \alpha_{00} & \beta_{00} \\ \overline{\tau_{01}} & \tau_{01} & \alpha_{01} & \beta_{01} \\ \overline{\tau_{10}} & \tau_{10} & \alpha_{10} & \beta_{10} \\ \overline{\tau_{11}} & \tau_{11} & \alpha_{11} & \beta_{11} \end{bmatrix}^{-1} \times \begin{bmatrix} K_{00} \\ K_{01} \\ K_{10} \\ K_{11} \end{bmatrix}$$

$C_0 := \tilde{C}_0 \| \sigma_C$
$C_1 := \tilde{C}_1 \| \overline{\sigma_C}$

for $a, b \in \{0,1\}^2$:
$c_{ab} := \kappa_{ab} \oplus \sigma_C \oplus \tau_{ab}$ *// encrypted output wire label color bit*
$d_{ab} := \widehat{\kappa}_{ab} \oplus (\alpha_{ab} \| \beta_{ab})$ *// encrypted α, β coefficients*

return $(C_0, C_1, (G, G', c_{00}, \ldots, c_{11}, d_{00}, \ldots, d_{11}))$

// INPUT: gate index idx; left wire label A^*; right wire label B^*; garbled gate info
// OUTPUT: output wire label C^*

$\underline{\mathsf{EvGate}(A^*, B^*, (G, G', c_{00}, \ldots, c_{11}, d_{00}, \ldots, d_{11}))}$:

$\chi_A := \mathrm{lsb}(A^*)$
$\chi_B := \mathrm{lsb}(B^*)$
$K^* \| \kappa^* \| \widehat{\kappa}^* := H(\mathsf{idx}, \chi_A, \chi_B; A^*, B^*)$
$\alpha^* \| \beta^* := d_{\chi_A, \chi_B} \oplus \widehat{\kappa}^*$
$C := K^* \oplus \alpha^* G \oplus \beta^* G'$
$\chi_C := c_{\chi_A, \chi_B} \oplus \kappa^*$
return $C \| \chi_C$

Figure 4: Our gate-hiding garbling scheme that uses only XOR

We reach a hybrid in which we have just changed any value of the form $H(\cdots \overline{W}_i^* \cdots)$ to be chosen uniformly. We consider a call to $\mathsf{GbGate}(\cdot, j; \cdots)$ for a downstream gate j such that $\max\{\mathsf{left}(j), \mathsf{right}(j)\} = i$. As before, we will argue that both the garbled gate and the inactive wire label $\overline{W}_j^*$ are being chosen uniformly at random.

In this call to GbGate, (C_0, C_1, G, G') are all being assigned as linearly independent linear combinations of $(K_{00}, \ldots, K_{11})$. Three of the four K_{ab} values are being chosen uniformly at random, so it makes no difference to choose G, G', and one of the C_z values (whichever one corresponds to $\overline{W}_j^*$) uniformly, then

solve for the other C_z value (the one that corresponds to W_j^*) by the honest evaluation process. Note that the honest evaluation process requires knowledge of coefficients $\alpha_{\chi_{\text{left}(i)},\chi_{\text{right}(i)}}, \beta_{\chi_{\text{left}(i)},\chi_{\text{right}(i)}}$. These changes cause G, G', and $\overline{W}_j^*$ to be chosen uniformly.

We can argue that the c_{ab} values of the garbled gate are chosen uniformly. The reasoning is the same as in the previous proof.

Finally, we consider the d_{ab} values of the garbled gate. Three of them are uniform because their corresponding masks $\widehat{\kappa}_{ab}$ values are being chosen uniformly. Changing them to be chosen explicitly at random means that the simulation no longer uses the associated α_{ab}, β_{ab} coefficients. Now the simulation only uses $(\alpha_{ab}, \beta_{ab})$ for a *single* a, b. By our reasoning in Section 5, the marginal distribution of these $(\alpha_{ab}, \beta_{ab})$ values is uniform. By this reasoning, the distribution of the fourth d_{ab} value is uniform. It makes no change to the distribution to choose this d_{ab} value first and then solve for $(\alpha_{ab}, \beta_{ab})$.

After all these changes, the entire garbled gate $(G, G', c_{00}, \cdots, c_{11}, d_{00}, \cdots, d_{11})$ and the inactive wire label $\overline{W}_j^*$ are explicitly being chosen uniformly.

The rest of the proof follows exactly the same reasoning as in the previous proof. □

5.4 Saving 4 Bits

As described, this construction requires $2\lambda + 12$ bits per gate. In this section we describe how it can be modified modified to require only $2\lambda + 8$ bits (conveniently making garbled gates a multiple of bytes in practice). The modification involves the distribution of α_i, β_i coefficients chosen by the garbler.

For a matrix like that in Equation 2, let $v \in \{0,1\}^4 \setminus \{0000, 1111\}$ denote its first column (the second column will always be its complement, $\overline{v}$). Let $\mathcal{D}_v$ denote the distribution over $\{\alpha_i, \beta_i\}$ coefficients used by the garbler. As discussed earlier, this distribution is expected to depend on v. For the scheme as described, the $\mathcal{D}_v$ distributions have support of 96 or 104 outcomes, depending on the parity of v.

It turns out that these distributions are "overkill" in some sense. The only properties we need are: (1) for every v, every outcome of $\mathcal{D}_v$ makes the corresponding matrix invertible, (2) for every i, the marginal distribution of (α_i, β_i) does not depend on v.

Later in Appendix A we describe a family of distributions with the following properties:

1. For all v, the distribution $\mathcal{D}_v$ assigns $(\alpha_1, \beta_1) = (0,0)$ with probability 1.
2. For every v and every $i \in \{2,3,4\}$, the marginal distribution on (α_i, β_i) induced by $\mathcal{D}_v$ is uniform in $\{01, 10, 11\}$.

Suppose we agree that the garbler will use these distributions. Then, because of the first property, there is no need for the garbler to send a ciphertext encrypting α_1, β_1. This saves a 2-bit ciphertext.

Then, since (α_2, β_2) are distributed uniformly over a set of 3 outcomes, we can use a row-reduction trick ([NPS99]) to remove the one-time-pad encryptions of α_2, β_2. The idea is to consider the cryptographic hash to have the form $H : \{0,1\}^* \to \{0,1\}^{\lambda+1} \times \underline{\mathbb{Z}_3}$, so that $H(A,B) = K \| \kappa \| \underline{\widehat{\kappa}}$, where $K \in \{0,1\}^\lambda$, $\kappa \in \{0,1\}$, and $\underline{\widehat{\kappa} \in \mathbb{Z}_3}$.

Now suppose we identify the set $\{01, 10, 11\}$ with $\mathbb{Z}_3$ and encrypt (α_i, β_i) via addition mod-3 with $\widehat{\kappa}_i$. If instead of choosing (α_2, β_2) uniformly, the garbler chooses $(\alpha_2, \beta_2) = -\widehat{\kappa}_2$, the resulting ciphertext would always be the zero ciphertext. The garbler can compute (α_2, β_2) in this way and then sample $\alpha_3, \beta_3, \alpha_4, \beta_4$ consistently — the result will be indistinguishable from before since $\widehat{\kappa}_2$ is pseudorandom as an output of H. Hence, there is again no need for the garbler to send anything for the encryption of (α_2, β_2). This saves the other two bits.

In summary,

- If the receiver is in case 1, then he/she uses coefficients $(\alpha_1, \beta_1) = (0,0)$

- If the receiver is in case 2, then he/she uses coefficients $(\alpha_2, \beta_2) = -\widehat{\kappa}_2$, where $\widehat{\kappa}_2$ is the one-time pad key obtained by calling H, interpreted as an element of $\mathbb{Z}_3 \cong \{01, 10, 11\}$.
- Otherwise, the receiver obtains (α_i, β_i) by decrypting the appropriate 2-bit ciphertext. In this case, we can encrypt the values with one-time pad in $\{0,1\}^2$.

The total size of the garbled gate becomes $2\lambda + 8$ (4 bits for the α_i, β_i coefficients and 4 bits for the output wire label's color bit).

References

[AF90] Martín Abadi and Joan Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2(1):1–12, 1990.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society Press, May 2013.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, October 2012.

[BL15] Mihir Bellare and Anna Lysyanskaya. Symmetric and dual PRFs from standard assumptions: A generic validation of an HMAC assumption. Cryptology ePrint Archive, Report 2015/1198, 2015. http://eprint.iacr.org/2015/1198.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 39–53. Springer, Heidelberg, March 2012.

[GAKS17] Daniel Günther, Ágnes Kiss, and Thomas Schneider. More efficient universal circuit constructions. Cryptology ePrint Archive, Report 2017/798, 2017. http://eprint.iacr.org/2017/798.

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 567–578. ACM Press, October 2015.

[HS] Wilko Henecka and Thomas Schneider. Memory efficient secure function evaluation. https://code.google.com/p/me-sfe/.

[KKS16] Carmen Kempka, Ryo Kikuchi, and Koutarou Suzuki. How to circumvent the two-ciphertext lower bound for linear garbling schemes. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT*, volume 10032 of *Lecture Notes in Computer Science*, pages 967–997, 2016.

[KKW16] W. Sean Kennedy, Vladimir Kolesnikov, and Gordon Wilfong. Overlaying circuit clauses for secure computation. Cryptology ePrint Archive, Report 2016/685, 2016. http://eprint.iacr.org/2016/685.

[KM11] Jonathan Katz and Lior Malka. Constant-round private function evaluation with linear complexity. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 556–571. Springer, Heidelberg, December 2011.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457. Springer, Heidelberg, August 2014.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

[KS16] Ágnes Kiss and Thomas Schneider. Valiant's universal circuit is practical. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 699–728. Springer, Heidelberg, May 2016.

[LMS16] Helger Lipmaa, Payman Mohassel, and Saeed Sadeghian. Valiant's universal circuit: Improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017, 2016. http://eprint.iacr.org/2016/017.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[MS13] Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 557–574. Springer, Heidelberg, May 2013.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139, New York, NY, USA, 1999. ACM.

[PSS09] Annika Paus, Ahmad-Reza Sadeghi, and Thomas Schneider. Practical secure evaluation of semi-private functions. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 89–106. Springer, Heidelberg, June 2009.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, Heidelberg, December 2009.

[TS] Stefan Tillich and Nigel Smart. Circuits of basic functions suitable for MPC and FHE. http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/.

[Val76] Leslie G. Valiant. Universal circuits (preliminary report). In Ashok K. Chandra, Detlef Wotschke, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA*, pages 196–203. ACM, 1976.

[WM17] Yongge Wang and Qutaibah m. Malluhi. Reducing garbled circuit size while preserving circuit gate privacy. Cryptology ePrint Archive, Report 2017/041, 2017. http://eprint.iacr.org/2017/041.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.

A Optimized Coefficients

Below is a description of coefficient distributions used in Section 5.4. These distributions were found using a brute-force search. For each v, we specify the distribution $\mathcal{D}_v$ over α_i, β_i values used in garbling. One can verify by inspection that these distributions satisfy the following properties:

- For all $v \in \{0,1\}^4 \setminus \{0000, 1111\}$, and all $(\alpha_2, \beta_2, \ldots, \alpha_4, \beta_4) \leftarrow \mathcal{D}_v$, the following matrix is invertible in $\mathbb{Z}_2$:
$$\begin{bmatrix} v_1 & \overline{v_1} & 0 & 0 \\ v_2 & \overline{v_2} & \alpha_2 & \beta_2 \\ v_3 & \overline{v_3} & \alpha_3 & \beta_3 \\ v_4 & \overline{v_4} & \alpha_4 & \beta_4 \end{bmatrix}$$
- For every v and every $i \in \{2,3,4\}$, the marginal distribution on (α_i, β_i) induced by $\mathcal{D}_v$ is uniform in $\{01, 10, 11\}$.

For $v \in \{0110, 1001\}$:

$$\begin{bmatrix} \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \\ \alpha_4 & \beta_4 \end{bmatrix} \leftarrow \left\{ \begin{array}{l} \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \\ \\ \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \end{array} \right\}$$

For $v \in \{0011, 1100\}$:

$$\begin{bmatrix} \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \\ \alpha_4 & \beta_4 \end{bmatrix} \leftarrow \left\{ \begin{array}{l} \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \\ \\ \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \end{array} \right\}$$

For $v \in \{0101, 1010\}$:

$$\begin{bmatrix} \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \\ \alpha_4 & \beta_4 \end{bmatrix} \leftarrow \left\{ \begin{array}{l} \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \\ \\ \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \end{array} \right\}$$

For $v \in \{0100, 1011\}$:

$$\begin{bmatrix} \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \\ \alpha_4 & \beta_4 \end{bmatrix} \leftarrow \left\{ \begin{array}{l} \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \\ \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \\ \\ \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \end{array} \right\}$$

For $v \in \{0010, 1101\}$:

$$\begin{bmatrix} \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \\ \alpha_4 & \beta_4 \end{bmatrix} \leftarrow \left\{ \begin{array}{l} \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \\ \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \\ \\ \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \end{array} \right\}$$

For $v \in \{0001, 1110\}$:

$$\begin{bmatrix} \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \\ \alpha_4 & \beta_4 \end{bmatrix} \leftarrow \left\{ \begin{array}{l} \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \\ \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}, \\ \\ \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \end{array} \right\}$$

For $v \in \{0111, 1000\}$:

$$\begin{bmatrix} \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \\ \alpha_4 & \beta_4 \end{bmatrix} \leftarrow \left\{ \begin{array}{l} \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array} \right\}$$